A Patchy Dynamic Programming Scheme for a Class of Hamilton-Jacobi-Bellman Equations*

Simone Cacace†, Emiliano Cristiani‡, Maurizio Falcone§, Athena Picarelli¶

July 19, 2011

Abstract

In this paper we present a new parallel algorithm for the solution of Hamilton-Jacobi-Bellman equations related to optimal control problems. The main idea is to divide the domain of computation into subdomains following the dynamics of the control problem. This results in a rather complex geometrical subdivision, but has the advantage that every subdomain is invariant with respect to the optimal controlled vector field, so that we can compute the value function in each subdomain assigning the task to a processor and avoiding the classical transmission condition on the boundaries of the subdomains. For this specific feature the subdomains are patches in the sense introduced by Ancona and Bressan in [1]. Several examples in dimension two and three illustrate the properties of the new method.

Keywords. patchy methods, Hamilton-Jacobi equations, parallel methods, minimum time problem, semi-Lagrangian schemes.

AMS. 65N55, 49L20

1 Introduction

The numerical solution of partial differential equations obtained by applying the Dynamic Programming Principle (DPP) to nonlinear optimal control problems is a challenging topic that can have a great impact in many areas, e.g. robotics,

*This work has been partially supported by AFOSR Grant no. FA9550-10-1-0029 and by the Grant SAPIENZA 2009 "Analisi ed approssimazione di modelli differenziali nonlineari in fluidodinamica e scienza dei materiali". The authors also wish to thank the CASPUR Consortium for its technical support

†Dipartimento di Matematica, SAPIENZA - Università di Roma, P.le Aldo Moro 2, 00185 Rome, Italy. simone.cacace@gmail.com

‡Dipartimento di Matematica, SAPIENZA - Università di Roma, P.le Aldo Moro 2, 00185 Rome, Italy. emiliano.cristiani@gmail.com

§(corresponding author) Dipartimento di Matematica, SAPIENZA - Università di Roma, P.le Aldo Moro 2, 00185 Rome, Italy. falcone@mat.uniroma1.it

¶Dipartimento di Matematica, SAPIENZA - Università di Roma, P.le Aldo Moro 2, 00185 Rome, Italy. athena.picarelli@gmail.com aeronautics, electrical and aerospace engineering. Indeed, by means of the DPP one can characterize the value function of a fully–nonlinear control problem (including also state/control constraints) as the unique viscosity solution of a nonlinear Hamilton–Jacobi equation, and, even more important, from the solution of this equation one can derive the approximation of a feedback control for the system (see the next section for more details). This result is the main motivation of a PDE approach to control problems and represents the main advantage over other methods, such as those based on the Pontryagin minimum principle, that gives only an open-loop control (see e.g. [27]). It is worth to mention that the characterization via the Pontryagin principle gives only necessary conditions for the optimal trajectory (the state) and optimal open-loop control (the co-state). Although, from the numerical point of view, this system can be solved via shooting methods for the associated two point boundary value problem, in real applications the initial guess to start with is a particularly difficult choice for the co-state and often requires many efforts and tentatives. This is why it can be rather efficient to combine the dynamic programming and the Pontryagin approaches and use the approximate value function in order to compute a suitable initial guess for direct methods as proposed in [21].

In this paper we mainly focus on the *minimum time problem*, which is associated to the following Hamilton-Jacobi-Bellman equation

$$\begin{cases} \max_{a\in A}\{-f(x,a)\cdot\nabla u(x)-1\}=0\,, & x\in\mathbb{R}^d\backslash\Omega_0 \\ u(x)=0\,, & x\in\Omega_0 \end{cases} \tag{1}$$

where d is the dimension of the state, $A\subset\mathbb{R}^m$ is a compact set defining the admissible controls, Ω_0 is the target set to be reached in minimal time and $f:\mathbb{R}^d\times A\to\mathbb{R}^d$ is the dynamics of the system. The value function $u:\mathbb{R}^d\to\mathbb{R}$ at the point x is the minimal time to reach the target starting from x ($u(x)=+\infty$ if the target is not reachable). For numerical purposes, the equation is solved in a bounded domain $\Omega\supset\Omega_0$. Boundary conditions on $\partial\Omega$ will be detailed later. Let us just mention that the same technique can be applied to more general control problems (see the last section for an example).

The techniques used to obtain a numerical approximation of the solution of equation (1) have been mainly based on Finite Differences [18, 34] and Semi-Lagrangian schemes [22, 25]. More recently, Finite Elements methods based on Discontinuous Galerkin approximations have been proposed, due to their ability to deal with non regular functions, which are the typical solutions appearing in the framework of viscosity solutions (see [17, 36, 16]). It is rather important to note that traditional approximation schemes presented for example in [22] and [18] are based on a fixed point iteration scheme, which computes the solution at each node of the grid at every iteration. Denoting by M the number of nodes in each dimension and considering that the number of iterations needed for convergence is of order $O(M)$, the total cost of this full-grid scheme is $O(M^{d+1})$. We easily conclude that this algorithm is very expensive when the state dimension is $d\geq 3$, although it is rather efficient for low dimensional control problems as shown in [22] (see also the book [25]).

The “curse of dimensionality” issue has been attacked from several directions and new techniques have been proposed to accelerate convergence and/or to reduce the memory allocation requirements. In [9] authors proposed an algorithm that allows to allocate only a small portion of the grid at every iteration. Another proposal to reduce the computational effort is the so-called Fast Marching method [33, 38]. While the full-size grid is always allocated, the computation is restricted to a small portion of the grid, thus saving CPU time. The cost of this method is of order $O(M^d \log M^d)$. In the original version, the Fast Marching method was derived for the Eikonal equation, corresponding (under a suitable change of variable) to equation (1) with $f(x, a) = a$ and $A = B_d(0, 1)$, the unit ball in $\mathbb{R}^d$ centred in 0 (see [20] for details). Despite the fact that the Fast Marching method is very efficient, at present its application to more general equations of the form $H(x, u(x), \nabla u(x)) = 0$ is not an easy task and it is still under investigation (see [12, 15, 19, 35]). For these reasons other methods have been proposed exploiting the idea that by applying the same method in a finite number of pre-defined directions one can accelerate convergence. These “sweeping” methods have been shown to be efficient for the eikonal equation [39] and, more recently, for rather general Hamiltonians [37].

A third possible strategy is based on the decomposition of the domain Ω. The problem is actually solved in subdomains Ω^j, $j = 1, \ldots, R$, whose size is chosen in order to reduce the number of grid nodes to a manageable size, see [26]. Therefore, rather than solving a huge problem in dimension d, one can solve R smaller subproblems working simultaneously on several processors. This produces a simple parallel algorithm. Depending on the choice of the subdomains Ω^j we can have some overlapping regions or a number of interfaces between the subdomains. The presence of interfaces and/or overlapping regions is a delicate point, since at each iteration of the algorithm it will be necessary to exchange information between processors to properly define the values on the interfaces. Without this communication load the result will not be correct. The interested reader can find in the book [32] a comprehensive introduction to domain decomposition techniques, whereas for an application to Hamilton-Jacobi equations we refer to [26, 13].

Finally, a decomposition of the domain based on the concept of “patchy feedbacks” has been proposed. The name “patchy” has been introduced in a work by Ancona and Bressan [1], where the authors studied the problem of the asymptotic stabilization of a control system. Their main result (see Theorem 2.1 in Section 2.3) states that, under suitable assumptions on the control system, stabilization can be obtained by means of a special feedback control, the *patchy feedback*, which is piecewise constant on a particular partition of the domain. Such a partition has the fundamental property that each part, or “patch”, is invariant with respect to the optimal dynamics driving the system. This result can be exploited from a numerical point of view. Indeed, once a patchy decomposition has been obtained, the computation can take advantage of the invariance of the subdomains assigning each patch to a processor, yielding a very efficient parallel algorithm that drops the need of communication between the processors. This is the spirit of what we call a “patchy method”. Unfortunately, the result of Ancona and Bressan is purely theoretical and their patchy decomposition turns out to be not constructive. Then, one has to face the problem of a numerical approximation of a dynamics invariant domain decomposition.

A first example of discrete patchy method has been proposed by Navasca and Krener in [30, 31]. The authors adopt a formal method developed by Al'brekht [4] that essentially translates the Hamilton-Jacobi-Bellman equation associated to a control problem into a system of algebraic equations, whose unknowns are the coefficients of the developments in power series of the cost and optimal feedback of the control problem. This gives an approximate solution in a small neighbourhood of the origin, which is the first patch of their domain decomposition. The solution is then extended in new patches around the first, by picking some boundary points from where optimal trajectories emanate (computed numerically backward in time). Those points define the centers of new neighbourhoods that can be used to restart the method. The solution is then obtained iteratively by fitting together the approximations in all the patches. More recently, it has been shown that this technique can be extended to obtain high-order accuracy in the regions where the value function is smooth (see [28] for more details).

Despite the high speed of the method, that actually does not use any grid, there are many open questions on its application. The first limitation is the strong regularity assumptions on the solutions necessary to set the problem in these terms. Indeed, it is well known that the most simple control problems may have optimal feedbacks which are not even continuous (see [27] and [5]). The second crucial point is the construction of the patchy decomposition that, in the examples contained in [30, 31], does not result in the invariance with respect to the optimal dynamics. This makes the solution rather inaccurate, especially near the boundaries of the patches.

The goal of this paper is to present and test a new patchy technique based on the coupling between a semi-Lagrangian scheme and a domain decomposition leading to a dynamic partition of the domain Ω into subdomains which are *invariant with respect to the optimal dynamics and to the optimal trajectories to the target*. To this end we will use the feedback controls computed by means of the semi-Lagrangian scheme on a rather coarse grid, then this information will be plugged into a dynamic algorithm which allows to obtain a domain decomposition where the domain boundaries correspond to approximate optimal trajectories. Finally, the computation of the value function on a fine grid is obtained via a parallel algorithm which assigns every sub-domain to a processor.

The paper is organized as follows. Section 2 is devoted to the general presentation of our problem, of the semi-Lagrangian scheme and of classical domain decomposition techniques for Hamilton-Jacobi equations. Moreover, we briefly describe the results on patchy methods which have been proved by Ancona and Bressan [1]. In Section 3 we present the patchy domain decomposition method and our algorithm to split the domain into invariant subdomains. We discuss there several issues related to the implementation of the method and its parallelization. Finally, Section 4 is devoted to the numerical tests on control problems in dimension 2 and 3.

2 Background

In this section we briefly introduce the numerical scheme we adopt to discretize equation (1) and the classical domain decomposition technique for Hamilton-Jacobi equations, including an algorithm that will be used in the following. Next, we recall the notion of patchy decomposition and the result by Ancona and Bressan concerning the asymptotic stabilization of control systems by means of patchy feedbacks, that inspired our patchy numerical method. We refer the interested reader to [1, 2, 3] for details.

2.1 The semi-Lagrangian scheme

Let us denote by $h > 0$ a fictitious time step (see the book [25] for details) and by $k > 0$ the space step. We introduce a structured grid G on Ω with nodes $x_i,\ i = 1, \dots, N$. We also denote by $\mathring{G}$ the internal nodes of G and by ∂G its boundary, whose nodes act as *ghost nodes*. We map all the values at the nodes onto a N-dimensional vector $U = (U_1, \dots, U_N)$. By a standard semi-Lagrangian discretization [6, 7, 22] of (1), it is possible to obtain the following scheme in fixed point form

$$U = F(U)\,, \tag{2}$$

where $F : \mathbb{R}^N \to \mathbb{R}^N$ is defined componentwise by

$$[F(U)]_i = \begin{cases} \min\limits_{a \in A} \{\mathbb{I}\,[U]\,(x_i + hf(x_i, a)) + h\} & x_i \in \mathring{G} \setminus \Omega_0\,, \\ 0 & x_i \in \Omega_0\,, \\ +\infty & x_i \in \partial G\,. \end{cases}$$

The discrete value function U is extended on the whole space Ω by a linear d-dimensional interpolation, represented by the operator $\mathbb{I}$ as described in [14]. We choose a variable time step $h = h_{i,a}$ such that $|h_{i,a}f(x_i, a)| = k$ for every $i = 1, \dots, N$ and $a \in A$, so that the point $x_i + h_{i,a}f(x_i, a)$ falls in one of the first neighboring cells. The minimum over A is evaluated by direct comparison, discretizing the set A with N_c points. Note that the definition $F(U) = +\infty$ on ∂G corresponds to impose state constraint boundary conditions. The final iterative scheme reads

$$U^{(n+1)} = F(U^{(n)})\,. \tag{3}$$

We refer to [22, 25] for details and convergence results. It is important to note that the scheme (2) produces an approximate feedback at every node of the grid. Moreover, the knowledge of the approximate value function at the nodes of the grid allows to extend it everywhere in Ω via interpolation. Then we can always obtain a feedback map $\Phi_h : \Omega \to A$ just defining

$$\Phi_h(x) := \arg\min_{a \in A} \{\mathbb{I}[U](x_i + hf(x_i, a)) + h\} \tag{4}$$

Under rather general assumptions (see [23]), it can be shown that this is an approximation of the feedback map constructed for the continuous problem as

$$\Phi(x) := \arg\max_{a \in A} \{-f(x,a) \cdot \nabla u(x) - 1\} \tag{5}$$

A detailed discussion on the construction of feedback maps via the value function is contained in [5] p. 140-143. It is important to note that weak convergence results apply also for Lipschitz continuous value functions.

2.2 Domain Decomposition method

The domain decomposition method allows to split the problem in Ω into R subproblems in subsets Ω^j, $j = 1, \dots, R$ such that $\Omega = \Omega^1 \cup \dots \cup \Omega^R$. For every pair $j \neq \ell$ of indexes corresponding to adjacent subdomains Ω^j and Ω^ℓ, let us denote by $\Omega^{j\ell}$ the non-empty overlapping zone $\Omega^j \cap \Omega^\ell$, which is assumed to contain at least one grid cell. This decomposition of the domain Ω induces a decomposition of the N degrees of freedom, so that to each subdomain Ω^j it is associated a number of nodes N^j. This allows to consider the restriction of U to Ω^j that we denote by U^j with $j = 1, \dots, R$. Similarly, we can define in a natural way the restriction to the subdomain Ω^j of the operator F in (2), denoted by $F^j : \mathbb{R}^{N^j} \to \mathbb{R}^{N^j}$. Then, we can define globally in Ω the following splitting operator $F_{\text{SPLIT}} : \mathbb{R}^N \to \mathbb{R}^N$, given componentwise by

$$[F_{\text{SPLIT}}(U)]_i \equiv \begin{cases} [F^j(U^j)]_i & \text{if } \exists j \text{ such that } x_i \in \Omega^j \setminus \bigcup_{\ell \neq j} \Omega^{j\ell}\,, \\ \min\limits_{j\,:\,x_i \in \Omega^j} \{[F^j(U^j)]_i\} & \text{otherwise}\,. \end{cases} \tag{6}$$

Following [26] it is easy to prove that fixed point iterations for F and F_{SPLIT} have the same solution.

We now describe a simple algorithm to compute the fixed point of F_{SPLIT}.

Domain Decomposition Algorithm:

Step 1. (Initialization) For $n = 0$ the initial guess $U^{(0)} \in R^N$ is fixed to 0 on the nodes corresponding to the target Ω_0 and $+\infty$ elsewhere.

Step 2. (Computation) $U^{(n+1/2)}$ is computed separately and in parallel in every subdomain Ω^j by

$$U^{j,(n+1/2)} = F^j(U^{j,(n)}) \qquad j = 1, \dots, R\,.$$

Step 3. (Coupling and synchronization among processors)

$$U_i^{(n+1)} = \begin{cases} U_i^{j,(n+1/2)} & \text{if } \exists j \text{ such that } x_i \in \Omega^j \setminus \bigcup_{\ell \neq j} \Omega^{j\ell}\,, \\ \min\limits_{j\,:\,x_i \in \Omega^j} \left\{U_i^{j,(n+1/2)}\right\} & \text{otherwise}\,. \end{cases} \tag{7}$$

Step 4. (Stopping criterion) If $\|U^{(n+1)}-U^{(n)}\|_\infty >$*toll* go to Step 2 with $n \leftarrow n+1$, otherwise stop.

In order to speed up the convergence of the algorithm above, we use iterations of Gauss-Seidel type, meaning that we employ the updated values of the nodes as soon as they are available. From now on the final solution computed by the Domain Decomposition algorithm will be denoted by U_{DD}.

2.3 Patchy feedbacks and stabilization for controlled systems

The concept at the basis of the numerical method we present in the next section is the notion of patch. It has been introduced for the first time by F. Ancona and A. Bressan [1], [2] in the context of the stabilization of controlled systems. Let us recall here for completeness their main definitions and results.

We consider the control system

$$\dot{y}(t) = f(y(t), a(t)) \qquad a(t) \in A, \tag{8}$$

assuming that the control set $A \subseteq \mathbb{R}^m$ is compact and the dynamics $f : \mathbb{R}^d \times A \to \mathbb{R}^d$ is sufficiently smooth. Moreover we choose as admissible controls all the functions $a(\cdot)$ belonging to

$$\mathcal{A} := \{a : (0,\infty) \to A : \ a(\cdot) \text{ is measurable}\} \ .$$

For every initial point $x \in \mathbb{R}^d$ and admissible control $a_0 \in \mathcal{A}$, we denote by $y(\cdot\,; x, a_0)$ the corresponding trajectory, which is an absolutely continuous function defined on some maximal interval $[0; \tau^{\max}(y))$, satisfying the system (8) for a.e. $t > 0$ with initial condition $y(0) = x$ and control a_0.

The following definition extends to control systems the classical notion of stability.

Definition 2.1 *The system (8) is said to be globally asymptotically controllable (to the origin) if the following holds:*

1. *for each $x \in \mathbb{R}^d$ there exists some admissible control a_0 such that the trajectory $t \to y(t) = y(t; x, a_0)$ is defined for all $t \geq 0$ and $y(t) \to 0$ as $t \to \infty$,*
2. *for each $\varepsilon > 0$ there exists $\delta > 0$ such that for each $x \in \mathbb{R}^d$ with $|x| < \delta$ there is an admissible control a_0 as in 1. such that $|y(t)| < \varepsilon$ for all $t \geq 0$.*

Given an asymptotic controllable system, a classical problem is to find a feedback control $a = k(y) : \mathbb{R}^d \to A$ such that *all the trajectories* of the corresponding closed loop system

$$\dot{y} = f(y, k(y)) \tag{9}$$

tend asymptotically to the origin.
Since this problem may not admit any solution in the class of *continuous feedbacks*, Ancona and Bressan introduce and investigate the properties of a particular class of *discontinuous feedbacks*, the so-called *patchy feedbacks*.

The following definition gives the fundamental concept of patch.

Definition 2.2 *Let be $\Omega \subset \mathbb{R}^d$ an open domain with smooth boundary $\partial\Omega$ and g a smooth vector field defined on a neighborhood of $\overline{\Omega}$. We say that the pair (Ω, g) is a patch if Ω is a positive-invariant region for g, i.e. at every boundary point $y \in \partial\Omega$ the inner product of g with the outer normal n satisfies*

$$\langle g(y), n(y) \rangle < 0.$$

Then, by means of a superposition of patches, we get the notion of patchy vector field on a domain $\Omega \subset \mathbb{R}^d$.

Definition 2.1 *We say that $g : \Omega \to \mathbb{R}^d$ is a patchy vector field if there exists a family of patches $\{(\Omega^\alpha, g^\alpha) : \alpha \in \mathcal{I}\}$ such that*

- *$\mathcal{I}$ is a totally ordered index set,*
- *the open sets Ω^α form a locally finite covering of Ω,*
- *the vector field g can be written in the form*

$$g(y) = g^\alpha(y) \qquad \text{if} \qquad y \in \Omega^\alpha \setminus \bigcup_{\beta > \alpha} \Omega^\beta.$$

We use $(\Omega, g, (\Omega^\alpha, g^\alpha)_{\alpha \in \mathcal{I}})$ to indicate the patchy vector field and the family of patches.

By applying the previous definitions to the closed loop system (9) we define a *patchy feedback* control as a piecewise constant map $k : \mathbb{R}^d \to A$ such that the vector field $g(y) := f(y, k(y))$ is a patchy vector field. More precisely:

Definition 2.2 *Let $(\Omega, g, (\Omega^\alpha, g^\alpha)_{\alpha \in \mathcal{I}})$ be a patchy vector field. Assume that there exist control values $k^\alpha \in A$ such that, for each $\alpha \in \mathcal{I}$*

$$g^\alpha(y) = f(y, k^\alpha) \qquad \forall y \in \Omega^\alpha \setminus \bigcup_{\beta > \alpha} \Omega^\beta.$$

Then the piecewise constant map

$$k(y) = k^\alpha \qquad \text{if} \qquad y \in \Omega^\alpha \setminus \bigcup_{\beta > \alpha} \Omega^\beta$$

is called a patchy feedback control on Ω.

Definition 2.3 *A patchy feedback control $k : \mathbb{R}^d \setminus \{0\} \to A$ is said to asymptotically stabilize the closed loop system (9) with respect to the origin if the following holds:*

1. *for each $x \in \mathbb{R}^d \setminus \{0\}$ and for every trajectory $y(\cdot)$ of (9) starting from x one has $y(t) \to 0$ as $t \to \tau^{\max}(y)$,*

2. *for each $\varepsilon > 0$ there exists $\delta > 0$ such that, for each $x \in \mathbb{R}^d \setminus \{0\}$ with $|x| < \delta$ and for every trajectory $y(\cdot)$ of (9) starting from x one has $|y(t)| < \varepsilon$, for all $0 \le t < \tau^{\max}(y)$.*

Finally, the main result of Ancona and Bressan can be summarized as follows.

Theorem 2.1 *If the system (8) is asymptotically controllable, then it admits an asymptotically stabilizing patchy feedback control.*

3 The patchy domain decomposition

In this section we introduce our new numerical method for solving equations of Hamilton-Jacobi-Bellman type. In particular we focus on the minimum time problem (1). The main feature of the new method is the technique we use to construct the subdomains of the decomposition, which are "patches" in a sense inspired by the definitions of the previous section. Indeed, we will see that these patches turn out to be quite invariant with respect to the optimal dynamics driving the system. Moreover, their boundaries can be rather complicated, but this is not a major difficult for the technique, since we do not need to apply any transmission condition between them.

Let us introduce two rectangular (structured) grids. The first grid should be rather *coarse* because it is used for preliminary (and fast) computations only. It will be denoted by $\widetilde{G}$ and its nodes by $\widetilde{x}_1, \dots, \widetilde{x}_{\widetilde{N}}$, where $\widetilde{N}$ is the total number of nodes. We will denote the space step for this grid by $\widetilde{k}$ and the approximate solution of the equation (1) on this grid by $\widetilde{U}_P$.

The second grid is instead *fine*, being the grid where we actually want to compute the numerical solution of the equation. It will be denoted by G and its nodes by $x_1, \dots, x_N$, where N is the total number of nodes ($N >> \widetilde{N}$). We will denote the space step for this grid by k and the solution of the equation (1) on this grid by U_P. We also choose the number R of subdomains (patches) to be used in the patchy decomposition and we divide the target Ω_0 in R parts denoted by Ω_0^j, with $j = 1, \dots, R$.
The patchy method can be described as follows.

Patchy Algorithm:

Step 1. (Computation on $\widetilde{G}$). We solve the equation on $\widetilde{G}$ by means of the classical domain decomposition algorithm described in Section 2.1. For coherence we choose the (static) decomposition made by R subdomains (as the number of patches). This leads to the function $\widetilde{U}_P$.

Step 2. (Interpolation on G). We define the function $U_P^{(0)}$ on the fine grid G by interpolation of the values $\widetilde{U}_P$. Then, we compute the approximate optimal control

$$\widetilde{a}^*(x_i) = \arg\min_{a\in A}\{\mathbb{I}[U_p^{(0)}](x_i + h_{i,a} f(x_i,a)) + h_{i,a}\}\,, \qquad x_i \in G. \tag{10}$$

Even if $\widetilde{a}^*$ is defined on G, we still use the symbol "tilde" to stress that optimal controls are computed using only coarse information. We delete $\widetilde{G}$.

Step 3. (Main cycle) For every $j = 1, \ldots, R$,

Step 3.1. (Creation of j-th patch). Using the (coarse) optimal control $\widetilde{a}^*$, we find the nodes of the grid G that have the part Ω_0^j of the target in their numerical domain of dependence. This procedure defines the j-th patch, naturally following the (approximate) optimal dynamics. This step will be detailed later in this section.

Step 3.2. (Computation in j-th patch). We initialize the j-th solution equal to $+\infty$ on the j-th patch and equal to 0 on the part Ω_0^j of the target (initial guess). Then, we apply iteratively the scheme (2) in the j-th patch until convergence is reached. Finally, the j-th solution is copied in the matrix that will contain the global solution U_P.

Details on Step 3.1. Following [11], the basic idea we adopt here is to divide the whole domain starting from a partition of the target only, letting the dynamics compute the partition in the rest of the domain. To this end we use the approximation of the optimal control given by $\widetilde{a}^*$ and then we obtain a domain decomposition fully compliant to the dynamics. More precisely, we divide the target Ω_0 in R parts, each associated to a colour indexed by a number $j = 1, \ldots, R$. Assume for instance that Ω_0 is a ball at the centre of the domain and focus on the subset of the target with a generic colour j, denoted by Ω_0^j, see Fig. 1(a). The goal is to find the subset of the domain which has Ω_0^j as numerical domain of dependence. To do that, we initialize the grid nodes with the values ϕ_i as follows

$$\phi_i = \begin{cases} 1\,, & x_i \in \Omega_0^j \\ 0\,, & x_i \in G\backslash\Omega_0^j \end{cases}\,, \qquad x_i \in G.$$

Then we solve the following *ad hoc* scheme, similar to the fixed-point algorithm (2) for the main equation

$$\phi_i = \mathbb{I}[\phi](x_i + h_i f(x_i, \widetilde{a}^*(x_i))) \,, \qquad x_i \in G. \tag{11}$$

Here h_i is chosen in such a way that $|h_i f(x_i, \widetilde{a}^*(x_i))| = k$. Once the computation is completed, the whole domain will be divided in three zones:

$$\Lambda_1^j = \{x_i : \phi_i = 1\} \,, \quad \Lambda_2^j = \{x_i : \phi_i = 0\} \,, \quad \Lambda_3^j = \{x_i : \phi_i \in (0,1)\} \,,$$

see Fig. 1(b). This is due to the fact that the interpolation operator $\mathbb{I}$ in the semi-Lagrangian scheme (11) mixes the values ϕ_i through a convex combination, thus producing values in $[0,1]$ even if the initial datum is in $\{0,1\}$. Since we need a sharp (non-overlapping) division of the domain, we "project" the colour j into a binary value

$$\phi_i^\sharp = \begin{cases} 1 \,, & \phi_i \geq \frac{1}{2} \\ 0 \,, & \phi_i < \frac{1}{2} \end{cases} \,, \qquad x_i \in G. \tag{12}$$

and then we define the subdomain $\Omega^j = \{x_i \in G \backslash \Omega_0^j : \phi_i^\sharp = 1\}$ as the j-th patch, see Fig. 1(c). Once all the patches $j = 1, \dots, R$ are computed, they are assembled together on the grid G. Thus the grid results to be divided in R patches, each associated to a different colour, as shown in Fig. 1(d).

The main point here is that patches Ω^j's are constructed to be invariant with respect to the optimal dynamics, meaning that the solution of the equation in each patch will not depend on the solution in other patches. This is equivalent to state that there is no crossing information through the boundaries of the patches.

We stress that Step 3.1 of the algorithm is not expensive, even if it is performed on the fine grid G. The reason for that is the employment of the pre-computed optimal control $\widetilde{a}^*$ in the equation (11), which avoids the evaluation of the minimum (see the scheme (2)). Moreover, the stop criterion for the fixed point iterations (11) can be very rough, since we project the colors at the end and then we do not need precise values.

Remark 3.1 (boundary conditions). To make it evident that each patch is independent of the others, we impose state constraint boundary conditions on the boundary of the patches. This kind of boundary conditions force the optimal direction $f(x, a^*)$ to point inside the patch. If patches are not invariant with respect to the optimal dynamics, this boundary condition leads to a huge error, which propagates all over the domain. It is important to note that, if the optimal trajectory runs along the boundary of a patch and the boundary of the patch is not aligned to the grid, the semi-Lagrangian scheme does not allow to select an optimal control which drives the dynamics exactly along the boundary, even if the set of admissible directions $f(x, A)$ include it. This is caused by the interpolation operator which necessarily makes use of some nodes outside the patch, where state constraints are found. The result is selecting an optimal

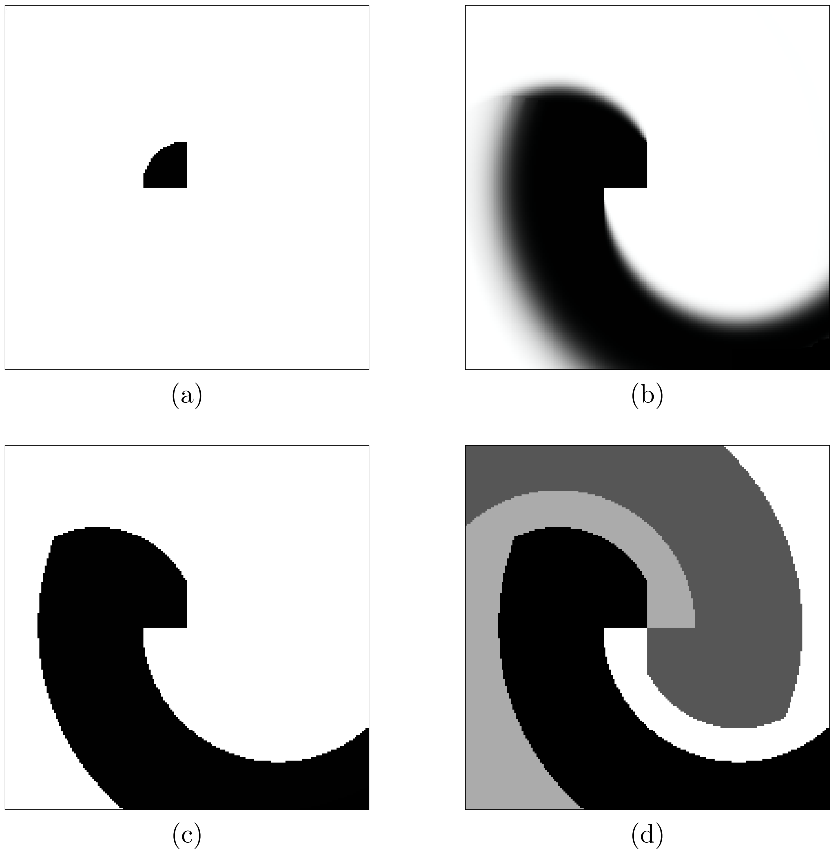

(a) (b) (c) (d)

Figure 1: Creation of patches for a test dynamics, R=4, Ω_0=small ball in the centre: (a) Select a subdomain Ω_0^j of the target Ω_0. (b) Find the nodes which depend, at least partially, on Ω_0^j. (c) Define Ω^j projecting the color in a binary value. (d) Assemble all patches.

direction which points toward a cell fully inside the patch (and not across two patches), see Fig. 3-(d) and its caption.

Remark 3.2 (patches as a partition of G). We have no guarantee that patches Ω^j's do not overlap nor that they cover the whole domain. Over the overlapping zones we can simply choose a colour at random. Instead, if they do not cover all the domain we can repeat the computation in the not-coloured nodes relaxing the condition in (12), i.e. choosing a different value for 1/2. Alternatively, in the case of isolated not-coloured points, we can assign to them the colour of their neighbours.

Remark 3.3 (equivalence of patches). Patches are not meant to be equivalent in terms of number of nodes, nor in term of CPU time needed to compute the solution in them. As we will see in the following, the difference in CPU time between the patches will be the key property that lets the patchy algorithm overcome the classical domain decomposition method. On the other hand, too large differences will lead to a deterioration of the performance.

Strategies for parallelization

The patchy algorithm can be parallelized in two ways.

Method 1: Patches are processed *one after the other* and only the computation in each patch is performed in parallel, assigning a batch of nodes among processors. This strategy gives priority to saving CPU time. Indeed, processors are active all the time. On the other hand, the full grid G must be allocated in RAM and be visible by all processors all the time.

Method 2: Patches are distributed among processors and computation of each patch is serial. This strategy gives priority to memory allocation issues. As patches are invariant with respect to the optimal dynamics, processors do not need to communicate and they can have a non-shared memory. Each processor works on its patch and once the computation is done it returns the result to the master thread. Note that if the number of processors equals the number of patches this procedure is not efficient in term of CPU time, because once a processor finished its job, it can not be assigned to another one and remains idle. Instead, if the number of processors is less than the number of patches, processors are better used because once one of them has finished its job, it can take another job until all the patches are covered.

Remark. All tests presented in this paper are performed implementing Method 1 and in the following we will always refer to this choice.

4 Numerical investigation in dimension two

In this section we first list the dynamics considered for the numerical tests. Then, we investigate the optimality of the patchy decomposition and the performance of the algorithm with respect to the classical domain decomposition.

Numerical tests were performed on a server Supermicro 8045C-3RB using 1 CPU Intel Xeon Quad-Core E7330 2.4 Ghz and 8×4 GB RAM running under Linux Gentoo operative system.

4.1 Choice of benchmarks

We will test the method described above against three minimum time problems of the form (1). The numerical domain is always $\Omega = [-2, 2]^2$.

Test 1 (Eikonal)

$$d = 2\,, \qquad f(x_1, x_2, a) = a\,, \qquad A = B_2(0,1)\,, \qquad \Omega_0 = B_2(0, 0.5).$$

Test 2 (Fan)

$$d = 2\,, \qquad f(x_1, x_2, a) = |x_1+x_2+0.1|a\,, \qquad A = B_2(0,1)\,, \qquad \Omega_0 = \{x_1 = 0\}.$$

Test 3 (Zermelo)

$$d = 2\,, \qquad f(x_1, x_2, a) = 2.1a + (2, 0)\,, \qquad A = B_2(0,1)\,, \qquad \Omega_0 = B_2(0, 0.5).$$

In Fig. 2 we show the patchy decomposition for the three dynamics described above in the case $R = 8$, $N_c = 32$, $\widetilde{N} = 50$ and $N = 100$. We also superimpose the optimal vector field $f(x, \widetilde{a}^*)$ to show that patches are indeed (almost) invariant with respect to the optimal dynamics. Only a few arrows cross from a patch to another. We note that patches cover the whole domain but they are not equivalent in terms of area, even if the target Ω_0 was divided in $R = 8$ equal parts to generate the decomposition.

4.2 Optimality of the patchy decomposition

Beside the discretization error due to the numerical algorithm, the patchy algorithm introduces another error, which will be denoted by E_P. Error E_P can be found comparing the solution of the patchy algorithm with that of the classical domain decomposition method based on the same numerical scheme,

$$E_P := U_P - U_{DD}.$$

This additional error is exclusively due to the fact that patches are not completely dynamics-invariant, because they are found using information computed on the coarse grid $\widetilde{G}$. It is plain that we could in principle compute an optimal patchy decomposition for the grid G working directly on the fine grid G, but this would cost as much as computing the solution U on G.

In the following we study the norms $\|E_P\|_1$ and $\|E_P\|_\infty$ as a function of the space step size $\widetilde{k}$ and k. We report the results for $R = 16$, which is the largest number of patches we tested. Note that error E_P necessarily increases as R increases because the number of boundaries increases. We recall that we impose state constraints boundary conditions at the boundary of each patch, fixing a large value of the solution outside the patch. In this way each computation is completely independent of the others and the quality of the final global solution U_P at the boundaries of the patches only depends on the degree of invariance of the patches with respect to the optimal dynamics.

Results for the three dynamics are shown in Tables 1, 2 and 3.

We see that the first line of each table reports in many cases unsatisfactory results, caused by the excessive roughness of the grid $\widetilde{G}$ (see the case $\widetilde{k}$=0.08, corresponding to $\widetilde{N}$=50). Even the case $\widetilde{k} = k$ (i.e. the grid is not refined at all)

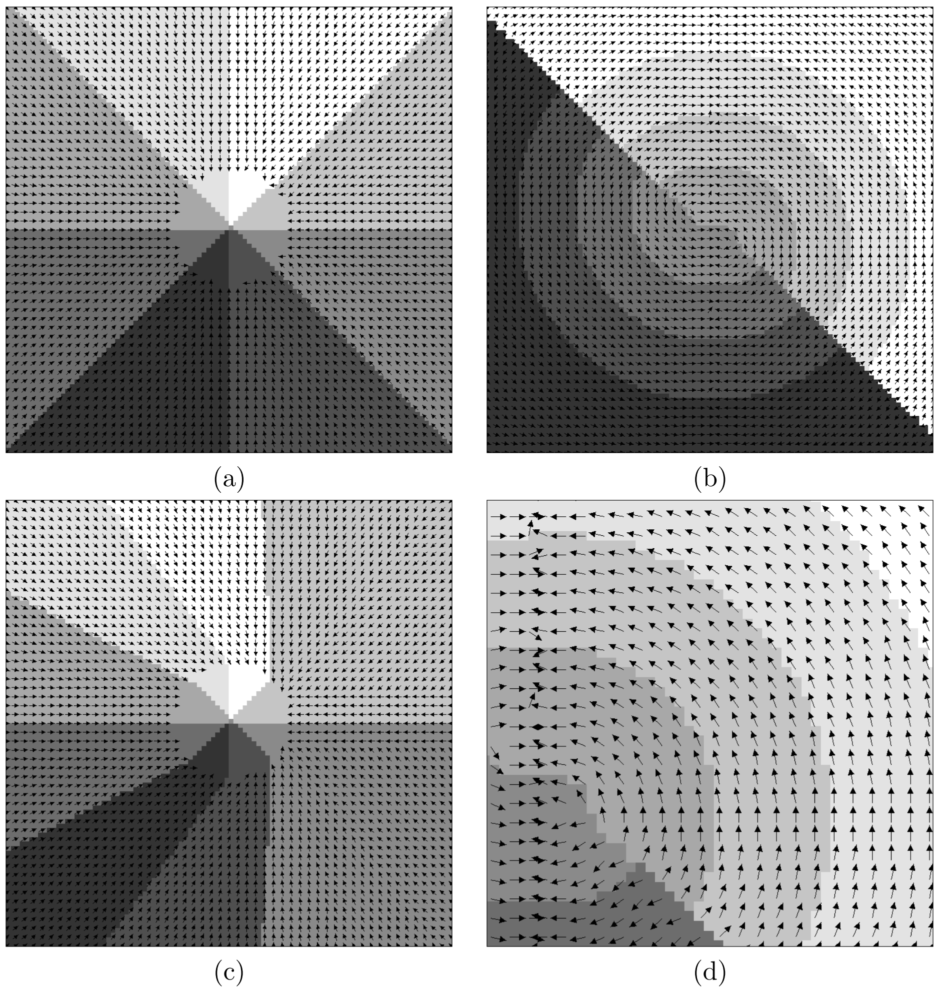


Figure 2: Patchy decompositions with $R = 8$, $N_c = 32$, $\widetilde{N} = 50$ and $N = 100$. Only few arrows are shown for clarity of visualization. (a) Eikonal, (b) Fan, (c) Zermelo, (d) a detail of (b).

is not satisfactory. This can be explained by recalling that the computations on the two grids are not equivalent because the second one employs the state constraint boundary conditions at the boundaries of each patch. If the grid is not fine enough, the error due to the boundary conditions is large, and tends to propagate inside each patch. Conversely, if $\widetilde{G}$ has at least 100 nodes per dimension ($\widetilde{k}$ $\geq$0.04), the behaviour of the error is surprisingly good because it decreases as k decreases (for any fixed $\widetilde{k}$) and $\|E_P\|_1$ is of the same order of k itself. Two comments about this behaviour are in order. First, the error caused

Table 1: Patchy error $\|E_P\|_1$ ($\|E_P\|_\infty$). Dynamics Eikonal, $N_c = 32$, $R = 16$

	$k = 0.08$	$k = 0.04$	$k = 0.02$	$k = 0.01$	$k = 0.005$
$\bar{k}$=0.08	0.436 (0.960)	0.275 (1.856)	0.102 (0.048)	0.065 (0.034)	0.048 (0.026)
$\bar{k}$=0.04	–	0.088 (0.046)	0.029 (0.023)	0.014 (0.042)	0.005 (0.008)
$\bar{k}$=0.02	–	–	0.038 (0.029)	0.012 (0.013)	0.004 (0.008)
$\bar{k}$=0.01	–	–	–	0.011 (0.016)	0.006 (0.010)
$\bar{k}$=0.005	–	–	–	–	0.004 (0.008)

Table 2: Patchy error $\|E_P\|_1$ ($\|E_P\|_\infty$). Dynamics: Fan, $N_c = 32$, $R = 16$

	$k = 0.08$	$k = 0.04$	$k = 0.02$	$k = 0.01$	$k = 0.005$
$\bar{k}$=0.08	1.393 (3.023)	0.123 (1.507)	0.037 (0.315)	0.017 (0.263)	0.011 (0.263)
$\bar{k}$=0.04	–	0.114 (1.502)	0.032 (0.149)	0.011 (0.095)	0.006 (0.095)
$\bar{k}$=0.02	–	–	0.032 (0.111)	0.011 (0.061)	0.004 (0.037)
$\bar{k}$=0.01	–	–	–	0.011 (0.079)	0.004 (0.037)
$\bar{k}$=0.005	–	–	–	–	0.004 (0.037)

Table 3: Patchy error $\|E_P\|_1$ ($\|E_P\|_\infty$). Dynamics: Zermelo, $N_c = 32$, $R = 16$

	$k = 0.08$	$k = 0.04$	$k = 0.02$	$k = 0.01$	$k = 0.005$
$\bar{k}$=0.08	0.171 (0.293)	0.159 (0.059)	0.097 (0.057)	0.026 (0.027)	0.006 (0.016)
$\bar{k}$=0.04	–	0.101 (0.063)	0.033 (0.041)	0.011 (0.023)	0.004 (0.016)
$\bar{k}$=0.02	–	–	0.039 (0.039)	0.012 (0.023)	0.004 (0.016)
$\bar{k}$=0.01	–	–	–	0.011 (0.020)	0.005 (0.015)
$\bar{k}$=0.005	–	–	–	–	0.004 (0.016)

by the state constraints boundary conditions stays close to the boundaries and does not propagate in the interior. Second, the error in localizing the patches is much less effective than the error caused by the numerical scheme. Let us investigate the latter point in more detail: If patches are not at all invariant with respect to the dynamics, as in the classical domain decomposition algorithm, we do not expect the error decreases as k decreases, because in this case the error is mainly due to the missing information from the boundaries rather than to the numerical scheme. If instead patches are perfectly independent of each other, the error decreases as k decreases, because the error is now only due to the numerical scheme. Our numerical tests show a behaviour much more similar to the second case, in which the error of the numerical scheme is leading with respect to that of the decomposition.

We also note that the L^∞ error is always larger than the L^1 error. Quite often we find a very small number of nodes with a large error near the boundaries of the patches, especially at those nodes where two patches and the target meet. This mainly affect the L^∞ error but not the L^1 error. Finally we note that the results are similar for the three dynamics, showing a good robustness even for highly rotating vector fields like that of Fan dynamics.

Fig. 3-(a,b,c) shows the functions E_P for the three tests. These results confirm that the error is concentrated along the boundaries of the patches. In the Eikonal and Zermelo case the error starts propagating from the target and increases as long as characteristics go away. In the Fan case, instead, the largest error is found where patches and target meet. Note that in the Eikonal case (a) no error is found where patches boundaries are aligned to the grid, since the interpolation makes no use of nodes belonging to different patches. Fig. 3-(d) shows a detail of the Fan decomposition along with the approximate optimal vector field computed by means of the final solution U_P. The effect of the state constraints is perfectly visible (arrows point unnaturally inward) confirming the fact that each patch is computed independently (see Remark 3.1).

Fig. 4 shows the value function U and its level sets for the Eikonal test. Here we see small perturbations where patches meet. It is interesting to note that we they not meet forming a discontinuity, but rather a hollow.

4.3 Comparison of CPU times

In this section we compare the patchy algorithm with the domain decomposition algorithm in terms of CPU time. Before that, we report in Table 4 the times (in seconds) for the single steps of the patchy algorithm. Times for the Step 3.1 are the most interesting ones because this step is expected to be the slowest one after Step 3.2 (main computation on the fine grid). Thus, time spent in Step 3.1 can completely neutralize the advantage we hope to get in the subsequent main computation. As we can see, Step 3.1 is much more costly than Steps 1 and 2, but not so much compared with the main computation.

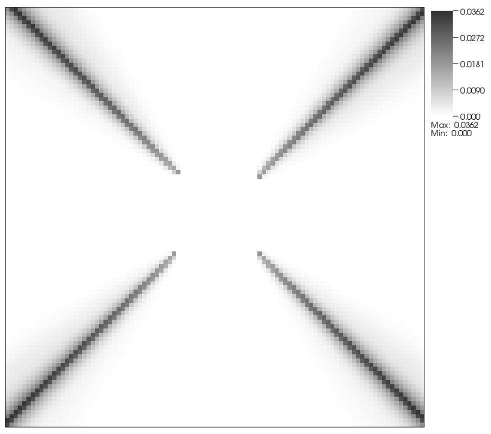
Tables 5, 6, 7 and 8 report the total CPU time (in seconds) for the three

Table 4: Processors: 4, N_c=32, Grid: $\widetilde{N} = 100^2 \to N = 800^2$, R=16

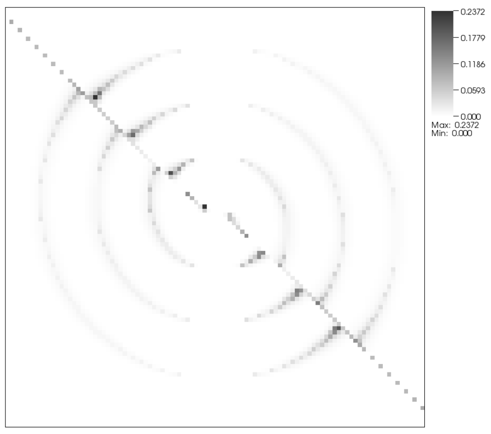
	Step 1	Step 2	Step 3.1 (all j's)	Step 3.2 (all j's)
Eikonal	2	1	23	409
Fan	2	2	52	796
Zermelo	2	1	30	512

dynamics of Sect. 4.1 as a function of the number of cores (1–4) and the number of patches (R=2,4,8,16). For the Eikonal test we also vary the number of discrete controls (N_c=16 and 32). These results are compared with the best outcome of the domain decomposition method obtained varying the number of domains (again 2,4,8,16)[1].

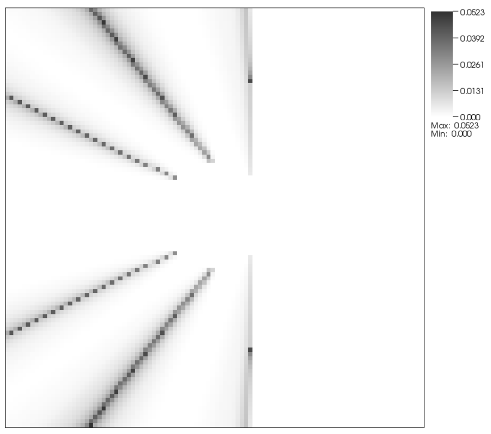
[1]The CPU time of the domain decomposition method does not vary a lot varying the number of domains, but small differences are present. They are due to the different order in which nodes are visited and synchronization overhead at the end of each iteration.



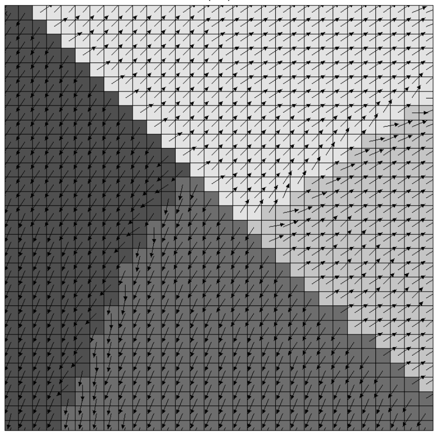
(a)



(b)



(c)

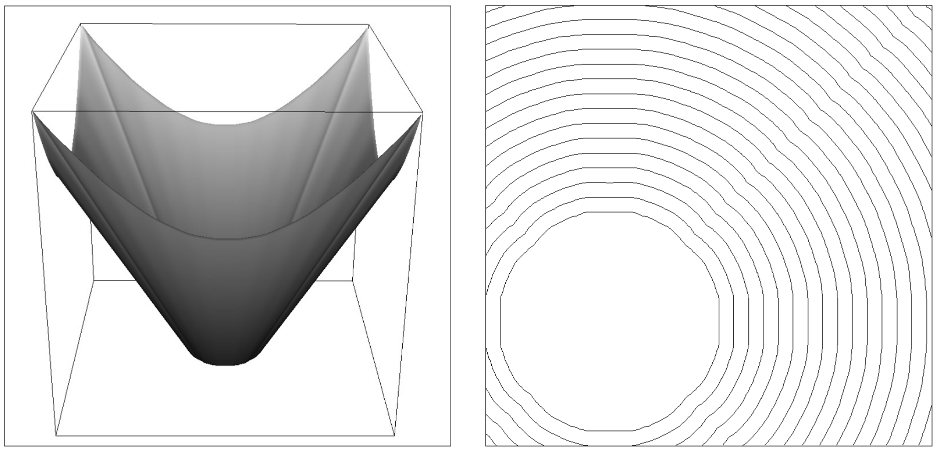


(d)

Figure 3: Patchy error E_P, $\widetilde{N} = 50 \to N = 100$ for (a) Eikonal, (b) Fan, (c) Zermelo. In (d) it is shown a detail of the patchy decomposition for the Fan dynamics, together with the optimal vector field $f(x, a^*)$ computed by means of the final patchy solution.

Table 5: Dynamics: Eikonal. N_c=16. Grid: $\widetilde{N} = 100^2 \to N = 800^2$

	2 domains	4 domains	8 domains	16 domains	Best DD
1 core	1547	1076	1058	933	1571
2 cores	845	595	574	504	820
4 cores	459	325	317	271	415



(a) (b)

Figure 4: Patchy solution for the dynamics Eikonal. (a) Value function and (b) a detail of its level sets. Small hollows are visible in correspondence of the lines $\{x = y\}$ and $\{x = -y\}$ as in Fig. 3-(a).

Table 6: Dynamics: Eikonal. N_c=32. Grid: $\widetilde{N} = 100^2 \to N = 800^2$

	2 domains	4 domains	8 domains	16 domains	Best DD
1 core	2702	1897	1843	1623	2785
2 cores	1462	998	968	872	1430
4 cores	771	532	514	435	716

First, we see that the speed-up with respect to the number of cores is very satisfactory and proves that the parallelization Method 1 (see Sect. 3) we implement here is sound. Second, we see that the CPU time decreases remarkably as the number of patches R increases. For R=16 the CPU time is largely less than that of the best domain decomposition method. This is one of the main results of the paper.

Differences among N_c=16 and N_c=32 are instead less clear, although the patchy algorithm should have an advantage for large N_c because of the smaller ratio between CPU time for Step 3.1 (one discrete control) and Step 3.2 (N_c discrete controls).

In order to understand why patchy algorithm is faster than domain decomposition method, let us consider again the Eikonal case with R=8, see Fig. 2-(a). If we visit the nodes in a single predefined order (i.e. we do *not* implement the Fast Sweeping technique [39] or similar ones), the eight subdomains need a different number of iterations to reach convergence. This is due to the fact that

Table 7: Dynamics: Fan. N_c=32. Grid: $\widetilde{N} = 100^2 \to N = 800^2$

	2 domains	4 domains	8 domains	16 domains	Best DD
1 core	3712	3322	3049	3172	4163
2 cores	2020	1746	1596	1559	2124
4 cores	1032	900	841	852	1069

Table 8: Dynamics: Zermelo. N_c=32. Grid: $\widetilde{N} = 100^2 \to N = 800^2$

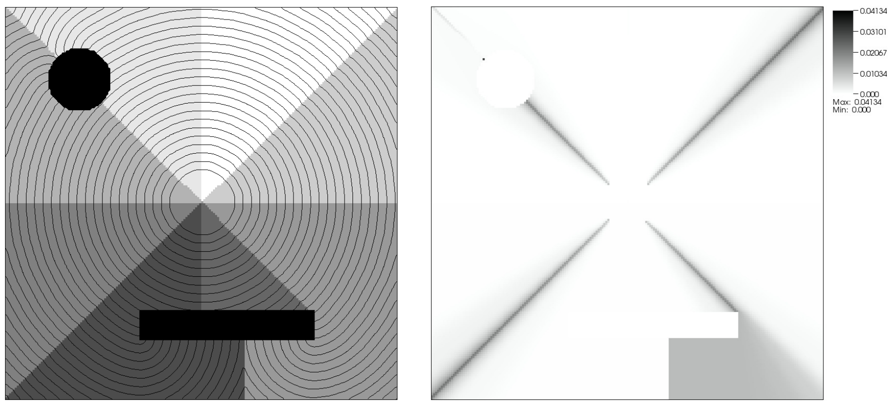
	2 domains	4 domains	8 domains	16 domains	Best DD
1 core	3113	2675	2126	2018	3209
2 cores	1651	1404	1111	1054	1640
4 cores	871	721	584	545	825

for some of them the visiting order corresponds to the upwind direction, while for the other domains the visiting order corresponds to the downwind direction. If we do not know *a priori* that the eight domains are invariant with respect to the optimal dynamics, we can not label as “done” the computation in a domain before computations in *all* domains are fully completed, because in any moment a new information can enter the domain, making necessary new computations. On the contrary, if we know *a priori* that domains do not depend on each other, we can label as “done” the computation in a domain as soon as the solution reached convergence, and use computational resources for other domains. Note that the more the number of domains the more efficient is the computation.

Finally note that usage of the Fast Sweeping technique can mitigate the performances of patchy method, but not neutralize them completely. Moreover, the patchy algorithm has the clear advantage that no synchronization nor crossing information among processors is needed. This is a great advantage when using distributed memory parallel computers, where communications are performed via cables connecting cluster nodes. This advantage is not really included in our experiments because our cores share a common RAM.

4.4 Patchy method with obstacles

We have also tried to run the patchy algorithm in a minimum time problem (Eikonal dynamics) with obstacles. In Fig. 5 we show the obstacles (the black circle and the rectangle), the level sets of the solution, the patchy decomposition and the patchy error E_P. The behaviour of the patchy decomposition is good because the dynamics drives the patches around the obstacles. If not influenced by the obstacles, the error is concentrated around the boundaries of the patches as expected. Instead, when a boundary meets an obstacles, the error can either stop propagating (see the circle) or spread out (see the rectangle, right side).



(a) (b)

Figure 5: Patchy domains bypass the obstacles driven by the dynamics. (a) Level sets of the value function and patchy decomposition. (b) Error E_P.

4.5 Limitations of patchy method

The overall efficiency of the patchy method depends on the dynamics and the shape of the target Ω_0. Unfortunately it is not always possible to reach a suitable patchy decomposition which allows to run the algorithm. This can happen for example if the target is very small and then it can not be divided in R subdomains.

Another issue, much more difficult to fix, comes out whenever there is a large difference between the sizes of the patches and possibly some of them degenerate in a subset of a few grid nodes. This is the case of the classical "Lunar Landing" problem

$$d = 2\,, \qquad f(x_1, x_2, a) = (x_2, a)\,, \qquad A = \{-1, 1\}\,, \qquad \Omega_0 = B_2(0, \varepsilon).$$

In this case the patchy decomposition consists of 2 large domains and $R - 2$ smaller domains, see Fig. 6-(a). The small domains degenerate to empty sets when ε tends to zero, because all the optimal trajectories tend to meet in only two switching lines.

A third dangerous case arises when some regions in Ω_0 are not reachable. If the dynamics make it impossible to reach the target from some point $x \in \Omega$, the value function is set to $u(x) = +\infty$. From the numerical point of view, the solution stays frozen at the value given as initial guess. At these points the optimal control (10) is not uniquely defined, and then the patchy decomposition can not be build. On the other hand, in the not-reachable regions the solution is in some sense already computed, then the issue can be easily fixed by a slight modification of the algorithm. After Step 1 we locate the regions where $\widetilde{U}_P$ is very large and then we do not consider those regions in the rest of computations.

4.6 Patchy decomposition for non-target problems

In the case of non-target problems we can not in principle build the patchy decomposition. Indeed, we recall that our patchy decomposition always starts from a decomposition of the target Ω_0. Nevertheless, in some special situations it is still possible to achieve the patchy decomposition, using some *a priori* knowledge on the solution of the problem. This is the case of the infinite horizon problem associated to the *linear-quadratic regulator*, studied by Krener and Navasca in [31] as a test for their patchy method (see also [30] and the Introduction for a brief description):

$$\min_{a\in\mathbb{R}} \int_0^\infty \left(\frac{1}{2}(y_1^2+y_2^2)+\frac{1}{2}a^2\right) dt \quad \text{subject to} \quad \begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = a \end{cases} \tag{13}$$

with $y_1(t=0)=x_1$ and $y_2(t=0)=x_2$. The exact value function for this problem turns out to be

$$u(x_1,x_2) = \frac{1}{2}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\begin{pmatrix} \sqrt{3} & 1 \\ 1 & \sqrt{3} \end{pmatrix}(x_1 \ x_2)$$

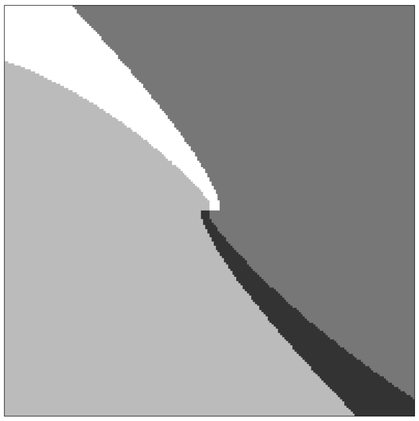
and it is easy to check that the origin $(0,0)$ is the only source of all characteristic curves. Then, using a small ball $B_2(0,\varepsilon)$ as a fictitious target, we are able to generate the patchy decomposition and then run the algorithm normally. In Fig. 6 we show the outcome of the simulation with the following parameters: $\Omega=[-1,1]^2$, $\varepsilon=0.05$, R=4, N_c=101, $A=[-3,3]$, $\widetilde{N}$=100, N=200. Note that the choice R=4 is due to limitations described in Section 4.5. Moreover, it is impossible to impose state constraints boundary conditions at the boundaries of the patches, since at some points the dynamics in (13) does not point inside the corresponding patch for any $a\in A$. To fix this, we imposed a Dirichlet boundary condition given by

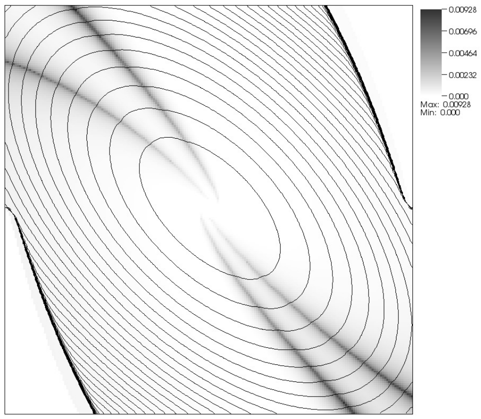
$$U_P|_{\partial\Omega^j} = U_P^{(0)}|_{\partial\Omega^j}\,, \qquad j=1,\ldots,R\,, \tag{14}$$

where $U_P^{(0)}$ is the first approximation of the solution computed in Step 2 of the patchy algorithm. Patchy errors are $\|E_p\|_1$=0.002 and $\|E_p\|_\infty$=0.009. Note that they are generally smaller than those in Tables 1-3 (computed for other dynamics) because of the more favorable boundary condition (14).

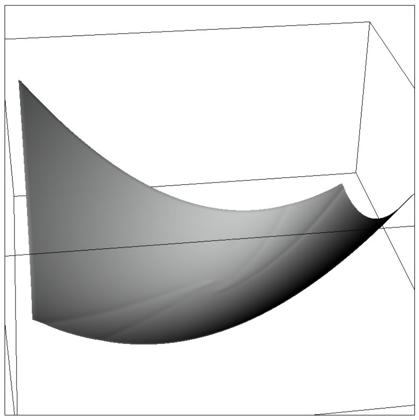
5 Patchy method's add-ons

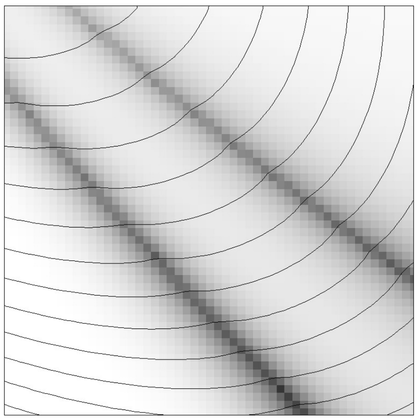
The patchy algorithm proposed in Section 3 has a multigrid nature, meaning that the computation of the solution on a rough grid is needed to start the optimal domain decomposition. Once this preliminary effort is done, it appears





(a) (b)





(b) (c)

Figure 6: (a) Decomposition in R=4 patches, (b) error function and level sets of the patchy solution (the solution is truncated at value $u = 3$ in order to remove the boundary effects which are very important for this dynamics.), (c) patchy solution U_P, (d) a detail of the error function shown in (b).

to be natural to use all the information we have collected in order to speed up the proposed algorithm. First of all, we will always impose by default the boundary condition (14) (note that it becomes available only after the computation on the rough grid). Further multigrid advantages we can take into account are listed in the following.

AO1. We use $U_P^{(0)}$ computed in Step 2 as initial guess for Step 3.2. In this way we save some iterations to reach convergence.

AO2. Before Step 3.2 we order the nodes belonging to each patch in such a way they fit as much as possible the *causality principle* [35]. For example we can order the nodes with respect to their values. This ordering is optimal if the characteristic lines coincide with the gradient lines of the solution, as it happens in the case of the Eikonal equation. In general this is not true, anyway this ordering is often not too far from the optimal one.

AO3. In Step 3.2 we reduce the number of discrete controls used in the numerical scheme, eliminating those controls which are "far" from the optimal one $\widetilde{a}^*(x_i)$ as computed by the first computation on the rough grid (Step 2). For example, if $A = B_2(0,1)$ we can introduce a reduction factor $r > 1$ and replace A with the set

$$A_r = \left\{ a \in A \ : \ a \cdot \widetilde{a}^* \geq \cos\left(\frac{\pi}{r}\right) \right\}.$$

This is the only add-on which introduces a new error in the solution, which is anyway negligible in most cases.

We point out that the patchy method can easily become an actual multigrid method. Indeed, we can in principle repeat the algorithm introducing a sequence of grids $G_1, G_2, \ldots$ one finer than the other, until the desired precision is reached.

In order to study the effect of the previously described add-ons, we introduce them one at a time and we compare the CPU time with the basic algorithm. Then, we apply all the features together. Results are reported in Table 9.

Table 9: Effects of add-ons. 2 procs, $R = 8$, $N_c = 32$. Controls reduced by factor 4

dynamics	grid size	basic	AO1	AO2	AO3	AO1+AO2+AO3
Eikonal	$100^2 \to 200^2$	20.0	19.2	9.6	9.1	5.7
Eikonal	$100^2 \to 400^2$	130.7	130.2	40.5	43.6	17.8
Eikonal	$100^2 \to 800^2$	928.1	924.6	238.8	298.1	100.6
Fan	$100^2 \to 200^2$	31.9	31.0	11.4	14.0	7.6
Fan	$100^2 \to 400^2$	209.8	205.7	43.5	72.3	20.6
Fan	$100^2 \to 800^2$	1571.9	1564.0	247.3	529.6	110.6
Zermelo	$100^2 \to 200^2$	23.2	22.6	11.5	10.7	6.7
Zermelo	$100^2 \to 400^2$	143.5	142.4	46.2	51.0	20.3
Zermelo	$100^2 \to 800^2$	1071.4	1057.9	290.1	345.5	111.3

Note that CPU times for this test are lower than those in Section 4.3 because of the more favorable boundary condition (14).

6 Numerical tests in dimension three

We solve three 3D minimal time problems of the form (1). The numerical domain is $\Omega = [-2,2]^3$ for all tests.

Test 1 (Eikonal 3D)

$$d = 3\,, \qquad f(x_1,x_2,x_3,a) = a\,, \qquad A = B_3(0,1)\,, \qquad \Omega_0 = B_3(0,0.5).$$

Test 2 (Fan 3D)

$$d = 3\,, \quad f(x_1,x_2,x_3,a) = |x_1+x_2+x_3+0.1|a\,, \quad A = B_3(0,1)\,, \quad \Omega_0 = \{x_1 = 0\}.$$

Test 3 (Brockett problem [8, 29])

$$d = 3\,, \quad f(x_1,x_2,x_3,a) = (a_1, a_2, x_1a_2 - x_2a_1)\,, \quad A = [-5,5]^2\,, \quad \Omega_0 = B_3(0,0.25).$$

For Tests 1 and 2 we used N_c=189 discrete controls uniformly distributed on the unit sphere, while for Test 3 we used only N_c=9 discrete controls in $\{-5,0,5\}^2$. The latter choice is motivated by the fact that using a larger number of discrete controls in $[-5,5]^2$ do not lead to a different result. This is expected because here we solve the minimum time problem associated to the Brockett dynamics, then the optimal strategy always requires to saturate the control to the maximal admissible value (5, in this case).

Results are reported in Table 10. Considering that for Tests 1-2 the number

Table 10: 3D tests. 4 procs, $R = 8$. Add-ons enabled (Eikonal and Fan: controls reduced by factor 4. Brockett: not reduced)

dynamics	grid size	CPU time	$\|E_P\|_1$	$\|E_P\|_\infty$
Eikonal 3D	$50^3 \to 100^3$	183	0.033	0.035
Eikonal 3D	$50^3 \to 200^3$	1217	0.029	0.042
Fan 3D	$50^3 \to 100^3$	165	0.064	0.187
Fan 3D	$50^3 \to 200^3$	1269	0.056	0.305
Brockett 3D	$50^3 \to 100^3$	132	0.229	0.024
Brockett 3D	$50^3 \to 200^3$	1557	0.165	0.020

of discrete controls is quite large, the CPU time is remarkable. Figure 7 shows the a level set of the value function for the Eikonal dynamics. It is perfectly visible the error located where the patches meet. Figure 8 shows the results for the Fan dynamics with 200^3 nodes. In Figs. 8(a)-(b) we show the boundaries of the patches and some level sets of the solution, respectively. Level sets should be plans, but the state constraints imposed by the computational box Ω bend them near $\partial\Omega$. In Fig. 8(c) we show some optimal trajectories to the target.

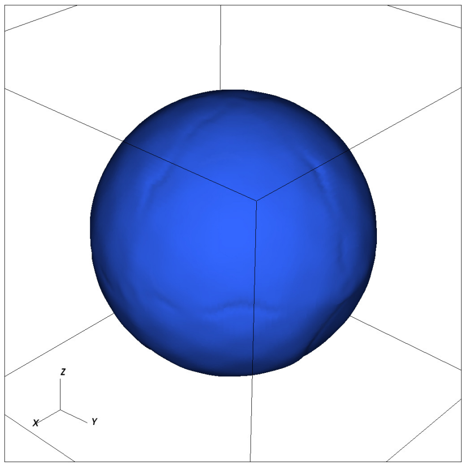


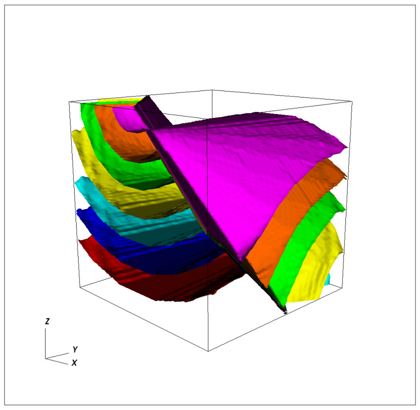
Figure 7: One level set of the value function for the Eikonal dynamics

Results for Brockett problem (Test 3) are different from the previous tests. First, CPU time turns out to be high with respect to the small number of discrete controls in use (just $N_c = 9$ controls). This could be related to the fact that characteristics are broken lines (see Figure 9(c)) that do not go directly to the target as in the Eikonal equation, nor bend slightly as for the Fan dynamics (see Figure 8(c)), but change direction istantaneously (see also control switch regions in Figure 9(b)), so that this dynamics takes much more time to move information through the domain. As a consequence, it increases the number of iterations the scheme needs to reach convergence and also the time to compute the patches. On the other hand, the patchy error in norm L^1 is larger than the error in norm L^∞ (see Table 10). This depends on the fact that the patchy decomposition obtained for this dynamics is rather complicate (see figure 9(a)), in particular patches arrange themselves in (suggestive) sets with very large boundary areas, which increase the number of nodes where we commit the patchy error E_P.

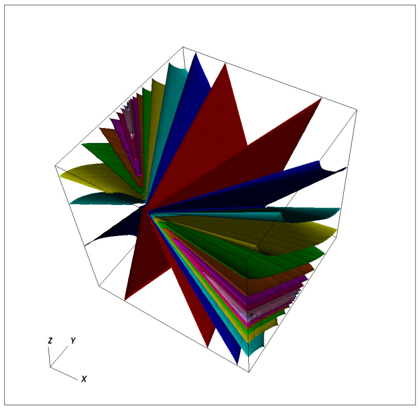
References

[1] F. Ancona and A. Bressan, *Patchy vector fields and asymptotic stabilization*, ESAIM: Control, Optimisation and Calculus of Variations, 4 (1999), 445–471.

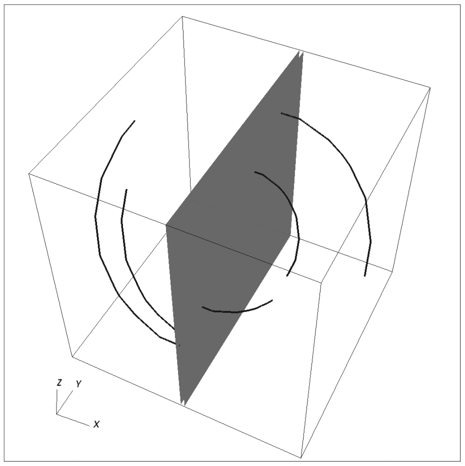
[2] F. Ancona and A. Bressan, *Flow stability of patchy vector fields and robust feedback stabilization*, SIAM J. Control Optim., 41 (2002), 1455–1476.



(a)



(b)



(c)

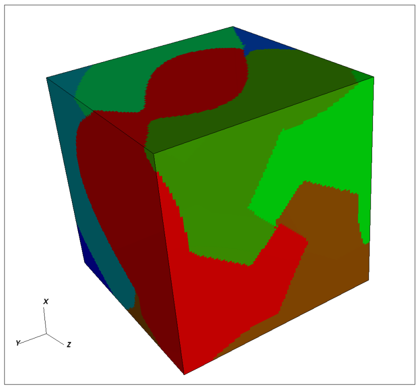
Figure 8: (a) Decomposition with 8 patches, (b) level sets of the solution, and (c) some optimal trajectories to the target

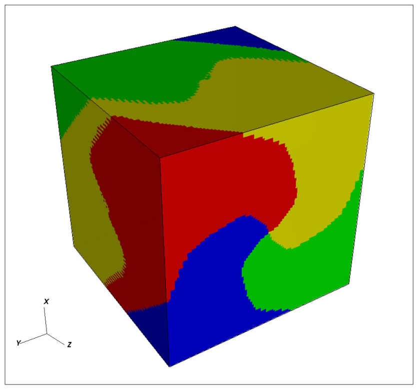
[3] F. Ancona and A. Bressan, *Nearly time optimal stabilizing patchy feedbacks*, Ann. I. H. Poincaré – AN, 24 (2007), 279–310.

[4] E. G. Al'brekht, *On the optimal stabilization of nonlinear systems*, J. Appl. Math. Mech., 25 (1961), 1254–1266.

[5] M. Bardi and I. Capuzzo Dolcetta, *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*, Birkhäuser, 1997.

[6] M. Bardi, M. Falcone, *An approximation scheme for the minimum time function*, SIAM Journal of Control and Optimization, 28 , 4, 1990, 950-965.


(a)


(b)

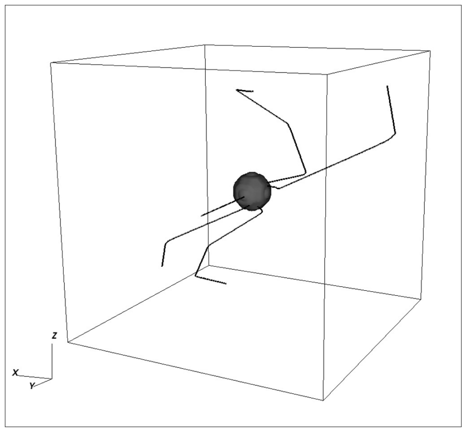

(c)

Figure 9: (a) Decomposition with 8 patches, (b) Regions with constant optimal controls, and (c) some optimal trajectories to the target

[7] M. Bardi, M. Falcone, *Discrete approximation of the minimal time function for systems with regular optimal trajectories*, in A. Bensoussan, J.L. Lions (eds.), *Analysis and Optimization of Systems, Lecture Notes in Control and Information Sciences*, n. 144, Springer-Verlag, 1990, 103-112.

[8] A. M. Bloch, M. Reyhanoglu, and N. H. McClamroch, *Control and stabilization of nonholonomic dynamic systems*, IEEE Trans. Aut. Cont., 37 (1992), 1746-1757.

[9] O. Bokanowski, E. Cristiani, and H. Zidani, *An efficient data structure and accurate scheme to solve front propagation problems*, J. Sci. Comput., 42 (2010), 251–273.

[10] A. Bressan, *Singularities of stabilizing feedbacks*, Rend. Sem. Mat. Univ. Pol. Torino, 56 (1998), 87–104.

[11] M. Breuss, E. Cristiani, P. Gwosdek, and O. Vogel, *An adaptive domain-decomposition technique for parallelization of the Fast Marching method*, Appl. Math. Comput., 218 (2011), 32-44.

[12] S. Cacace, E. Cristiani, and M. Falcone, *A local ordered upwind method for Hamilton-Jacobi and Isaacs equations*, Proceedings of 18th IFAC World Congress 2011.

[13] F. Camilli, M. Falcone, P. Lanucara, and A. Seghini, *A domain decomposition method for Bellman equations*, in D. E. Keyes and J. Xu (eds.), Domain Decomposition methods in Scientific and Engineering Computing, Contemporary Mathematics n.180, AMS, 1994, 477–483.

[14] E. Carlini, M. Falcone, and R. Ferretti, *An efficient algorithm for Hamilton-Jacobi equations in high dimension*, Comput. Visual. Sci., 7 (2004), 15–29.

[15] E. Carlini, M. Falcone, N. Forcadel, and R. Monneau, *Convergence of a Generalized Fast Marching Method for an Eikonal equation with a velocity changing sign*, SIAM J. Numer. Anal. 46 (2008), 29202952.

[16] Y. Chen and B. Cockburn, An adaptive high-order discontinuous Galerkin method with error control for the Hamilton-Jacobi equations. I. The one-dimensional steady state case. J. Comput. Phys. 226 (2007), no. 1, 10271058.

[17] Y. Cheng and C.W. Shu, *A discontinuous Galerkin finite element method for directly solving the Hamilton-Jacobi equations*, J. Comput. Phys. 223 (2007), no. 1, 398415.

[18] M.G. Crandall and P.L. Lions, *Two approximation of solutions of Hamilton-Jacobi equations*, Mathematics of Computation, 43, 1984, 1-19.

[19] E. Cristiani, *A fast marching method for Hamilton-Jacobi equations modeling monotone front propagations*, J. Sci. Comput., 39 (2009), 189–205.

[20] E. Cristiani and M. Falcone, *Fast semi-Lagrangian schemes for the Eikonal equation and applications*, SIAM J. Numer. Anal., 45 (2007), 1979–2011.

[21] E. Cristiani and P. Martinon, *Initialization of the shooting method via the Hamilton-Jacobi-Bellman approach*, J. Optim. Theory Appl., 146 (2010), 321–346.

[22] M. Falcone, *Numerical solution of dynamic programming equations*, Appendix A in [5].

[23] M. Falcone, *Some remarks on the synthesis of feedback controls via numerical methods*, in J.L. Menaldi, E. Rofman, A. Sulem (eds), Optimal Control and Partial Differential Equations, IOS Press, 2001, 456-465.

[24] M. Falcone, *Numerical methods for differential games based on partial differential equations*, International Game Theory Review, 8 (2006), 231–272.

[25] M. Falcone and R. Ferretti, *Semi-Lagrangian approximation schemes for linear and Hamilton-Jacobi equations*, SIAM, to appear.

[26] M. Falcone, P. Lanucara, and A. Seghini, *A splitting algorithm for Hamilton-Jacobi-Bellman equations* Applied Numerical Mathematics, 15 (1994), 207–218.

[27] W.H. Fleming and R. W. Rishel, Raymond, *Deterministic and stochastic optimal control.*, Series Applications of Mathematics, No. 1. Springer-Verlag, Berlin-New York, 1975.

[28] T. Hunt, *A proof of the higher order accuracy of the patchy method for solving the Hamilton-Jacobi-Bellmamn equation.* Ph.D. Thesis, University of California, Davis, 2011.

[29] K.A. Morgansen and R.W. Brockett,*Nonolonomic control based on approximate inversion*, Proceedings of the 1999 American Control Conference, 3515-3519.

[30] C. Navasca and A. J. Krener, *Patchy solutions of Hamilton-Jacobi-Bellman partial differential equations*, in A. Chiuso et al. (eds.), Modeling, Estimation and Control, Lecture Notes in Control and Information Sciences, 364 (2007), 251–270.

[31] C. Navasca and A. J. Krener, *The patchy cost and feedback for the HJB PDE*, Proceedings of the 18th International Symposium on Mathematical Theory of Networks and Systems (Blacksburg VA, USA), 2008.

[32] A. Quarteroni, A. Valli, *Domain decomposition methods for partial differential equations*, Oxford University Press, 1999.

[33] J. A. Sethian *A fast marching level set method for monotonically advancing fronts*, Proc. Natl. Acad. Sci. USA, **93** (1996), 1591–1595.

[34] J. A. Sethian, *Level set methods and fast marching methods*, Cambridge University Press, 1999.

[35] J. A. Sethian and A. Vladimirsky, *Ordered upwind methods for static Hamilton-Jacobi equations: theory and algorithms*, SIAM J. Numer. Anal., 41 (2003), 325–363.

[36] C.W. Shu, *High order numerical methods for time dependent Hamilton-Jacobi equations*, Mathematics and computation in imaging science and information processing, 4791, Lect. Notes Ser. Inst. Math. Sci. Natl. Univ. Singap., 11, World Sci. Publ., Hackensack, NJ, 2007.

[37] Y. Tsai, L. Cheng, S. Osher, H. Zhao, *Fast sweeping algorithms for a class of Hamilton-Jacobi equations*, SIAM J. Numer. Anal. 41 (2004), 673-694.

[38] J. N. Tsitsiklis, *Efficient algorithms for globally optimal trajectories*, IEEE Trans. Autom. Control, 40 (1995), 1528–1538.

[39] H. Zhao, *A fast sweeping method for Eikonal equations*, Math. Comp., 74 (2005), 603–627.